Reduction of wasted energy in a volunteer computing system through Reinforcement Learning

A. Stephen McGough[a,*], Matthew Forshaw[b]

[a]*School of Engineering and Computing Sciences, Durham University, DH1 3LE, U.K.*
[b]*School of Computing Science, Newcastle University, NE1 7RU, U.K.*

Abstract

Volunteer computing systems provide an easy mechanism for users who wish to perform large amounts of High Throughput Computing work. However, if the Volunteer Computing system is deployed over a shared set of computers where interactive users can seize back control of the computers this can lead to wasted computational effort and hence wasted energy. Determining on which resource to deploy a particular piece of work, or even to choose not to deploy the work at the current time, is a difficult problem to solve, depending both on the expected free time available on the computers within the Volunteer computing system and the expected runtime of the work – both of which are difficult to determine a-priori. We develop here a Reinforcement Learning approach to solving this problem and demonstrate that it can provide a reduction in energy consumption between 30% and 53% depending on whether we can tolerate an increase in the overheads incurred.

Keywords: Volunteer Computing, Energy, Reinforcement Learning, Task Scheduling

2014 MSC: 00-01, 99-00

*Corresponding author
Email addresses: stephen.mcgough@durham.ac.uk (A. Stephen McGough), m.j.forshaw@newcastle.ac.uk (Matthew Forshaw)

1. Introduction

Volunteer computing represents a powerful paradigm allowing organisations to exploit existing computational power, either owned by themselves or others, in order to solve large computational problems. It makes use of the idle time on computers in order to progress the computation, relinquishing control back to the normal user when they require it. Examples of volunteer computing systems include HTCondor (formerly known as Condor) [1], which is most often used by organisations who wish to exploit the idle time of computers which they already own, or BOINC [2], which is commonly used in situations where the organisation wishes to exploit computer power not owned by the organisation.

All volunteer computing systems need to handle the eviction of work due to a computer's user requiring the computational power back for their own use or due to system crashes and reboots. In the best case scenario the pieces of computational work distributed out are small enough that control can be passed back without the loss of any significant amount of work. If the interruption is expected to be short it may be possible to suspend the work – removing processing time from the work – subject to a threshold after which the work is evicted, thus saving energy from re-execution. This is not possible in the case of system crashes or reboots and if the user's active period is long can significantly increase the overheads on the execution time of the work. A third option is to use checkpoint and migration of work [3], though this is only possible for some types of work and on certain types of operating systems and may not lead to energy-efficient use of the system [4, 5]. It is also possible to develop your own checkpoint and migration mechanism bespoke to a particular application. More often than not, unless the organisation wishing to perform the work is willing to expend effort, the work will just be terminated and the current execution lost, thus directly leading to wasted energy. This can be exacerbated further if the work is repeatedly evicted.

Thus our problem can be summarised as determining the most appropriate time and best computer on which to schedule a particular piece of work in

order to maximise the chances of it running to completion. Given that we are not able to determine *a-priori* the next time a user will wish to make use of their computer, nor the execution time of a particular piece of work, this is a non-trivial problem to solve. As a secondary problem we aim to minimise energy consumption thus steering work towards more energy efficient computers where appropriate. We investigate this problem in the context of a volunteer computing system based within a single large organisation – such as a university HTCondor system.

In previous work [6] we have shown that it would not be feasible to restrict the execution times of submitted work in order to minimise evictions – showing that work execution times would need to be less than two minutes to ensure 95% of work would complete without eviction. Here we seek to better place work onto resources, or potentially choose not to deploy work onto a resource at a given time, to minimise the chance of eviction whilst at the same time attempting to place work on the most energy efficient computers.

In a large organisation there will be general trends for the times that computers will be used and when work will be submitted to the Volunteer computer system. However, the patterns which emerge are likely to be complicated and bespoke to the particular organisation. Detailed analysis could be performed in order to determine these pattern, though this would have little benefit for other organisations and would be quickly invalidated if usage patterns change – which is to be expected.

Reinforcement Learning [7] is a machine learning technique which is capable of adapting behaviour in order to maximise a given reward function. It has the advantages that it does not require initial training data and can constantly re-train itself to the changing environment. At each decision epoch one of two polices can be used. The first policy, often referred to as *exploitation*, is to select the action to perform which given previous evidence would seem to maximise the return. Continual use of this policy would lead to a non-adaptive solutions as actions which previously gave poor return would not be considered. Therefore the second policy, often referred to as *exploration*, is used to allow adaption

to the changing environment by selecting the action randomly. This does have the disadvantage that sub-optimal actions may be selected, however, it has the advantage that an action which may now gives good rewards can be tried allowing it to be used through exploitation in the future. It is therefore essential to get the right balance between these two polices – too greedy will lead to poor adaption to a changing environment whilst too explorative will lead to many bad action choices. On completion of an action a reward value is computed indicating how good the choice was to select this action and the action history is updated with this reward, increasing the reward history if the chosen action was good and decreasing the reward history if bad.

In this work we propose and evaluate the use of a Reinforcement Learning technique in order to 'learn' the patterns of the system. By employing a Reinforcement Learning technique which allows for both *exploration* and *exploitation* we are able to develop a system which is capable of both learning the particular patterns of a given Volunteer Computer system but also to adapt as the system changes.

The rest of this paper is set out as follows. We formally describe our Volunteer Computing model in Section 2 before performing an analytical analysis of historical trace-logs from our University based HTCondor system in Section 3. In Section 4 we discuss how we use a Reinforcement Learning approach to reduce the energy consumption for scheduling jobs within this system. Section 5 describes the environment which we will be simulating along with the Cluster simulation software used. We discuss related work in Section 6. Results from our simulation are presented in Section 7 before we present conclusions and future directions in Section 8.

2. Cluster Model

We explore the use of Reinforcement Learning for resource selection within a shared use computer system in which High Throughput Computing (HTC) jobs (we equate jobs with work and adopt this term hereafter) are run on the same

computers as comprise a set of open access clusters. Each cluster comprises of a set of computers geographically located within the organisation. Figure 1 illustrates the overall architecture of the computer system we are modelling.

Computers within a cluster are assumed to be under their own power management. In previous work we have shown how we can modify this policy in order to effect the energy consumption of the computers [8]. However, for this work we assume that the energy policy of the computers is not under our control. Instead we focus on how we can effect the energy consumption of the High Throughput Computing jobs. The High Throughput Computing System is under its own policy controlling such factors as how long after a user logs out of a computer it can be used for jobs and the selection policy for which computer to use. Our system also provides a mechanism by which the HTC system can wake up sleeping computers if required to perform work.

Interactive users, who are assumed to be able to log into any computer within the organisation, can arrive at any time that a particular cluster room is open and log into the computer of their choice. By contrast, high-throughput jobs are submitted through one centrally controlled access point with the system itself determining the computers that will be used. We possess trace data for both user types. We assume computers may go to sleep based on a pre-defined policy and that interactive users can always wake up a computer. Policies also allow

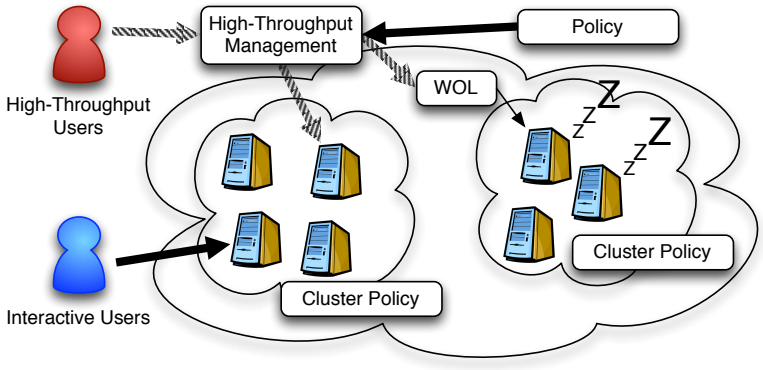

Figure 1: Overall architecture of multi-use Cluster

the high-throughput jobs to wake up the computers when needed.

Computers within the organisation can be in one of four states, those of sleeping, idle, User and HTC (Figure 2). The sleeping state equate to the Advanced Configuration and Power Interface (ACPI) specification [9] state S3, whilst all other states equate to ACPI state S0. It is assumed that computers will consume energy at a set rate in each state with User and HTC consuming energy at the same rate. We acknowledge that the rate of energy consumption would vary based on the individual workload in each state, however, as our intention here is to determine if energy could be saved by using Reinforcement Learning we can safely ignore this effect as it would become a scaling factor to our actual results.

As interactive users are the primary reason for the computers they take precedence. A user is able to log into a sleeping computer or an idle computer and if they log into a computer servicing a HTC job then the job will be evicted. Slight delays while a computer resumes from sleep or while a high-throughput job terminates are considered tolerable. Only idle computers can be used for HTC jobs. If it is desired to use a sleeping computer for a HTC job it first

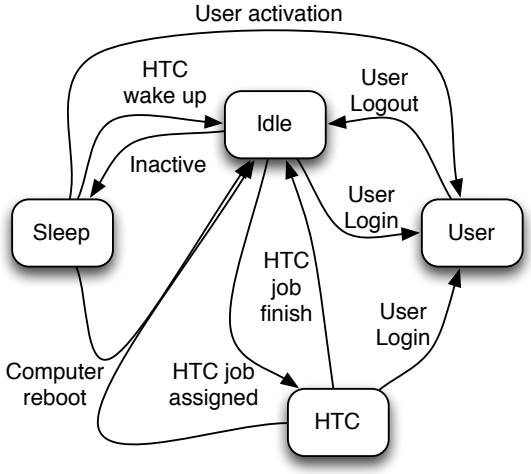

Figure 2: States and state transitions for computers

transitions into the idle state before it can be assigned a HTC job. This simplifies the wakeup procedure for computers which cannot determine if they are woken due to a computer reboot or HTC request. Computers are rebooted nightly from all states apart from User in which case the user remains active and the reboot is cancelled. All reboots return the computer to the idle state.

By contrast HTC jobs can be in one of two states, active or queued. Active jobs are being processed on a computer whilst queued jobs are inactive and waiting for a computer. In the event of a job being evicted from a computer due to a reboot or user login then it is placed back into the queue of pending jobs for re-scheduling to a computer. Thus a job may be stated and evicted from a computer several times before it completes execution. This situation is exacerbated by the fact that a HTC user may submit broken jobs which never complete.

We do not concern ourselves here with broken jobs which fail immediately or ones which finish in a short time. The first case have little if any energy requirements whilst it is not possible to distinguish the second case from non-broken jobs. For the purpose of this work all jobs which terminate within a finite amount of time are considered to be good. The difficulty is in distinguishing between bad jobs which are not going to terminate and good jobs which have just been unfortunate in their allocation to resources. In previous work [6] we identified these ambiguous jobs as *'miscreant'* and investigated techniques which could be used to classify these miscreant jobs as either good or bad. This allowed us to mark the jobs as bad and cease attempting to run them thus saving energy. However, this approach did not take into account the time and energy spent running a miscreant job before either good job completion of identifying it as a bad job. This work could also lead to false-positives where good jobs were identified as bad leading to a loss of productive work.

3. Analysis of a real HTC system

In this section we analyse the characteristics of our HTCondor system, based on our logs from 2010, in order to identify potential scenarios under which it would not be appropriate to run a job. We can bin jobs by the number of hours execution they require to complete. Figure 3 shows the percentage of jobs each day which required y hours of execution time – ignoring any time wasted through evictions. Note that this does not include jobs which failed to terminate as these jobs do not have a meaningful execution time. Most *'good'* jobs have an execution time less than three hours. However, there are a number of anomalies. Thus it is not safe to assume any job which has received over y hours of service will automatically be a *'bad'* job.

Although we do not have a prediction on the amount of time a job will take to execute we can use historical information from previous evicted executions in order to provide a lower bound for the execution time of a job. Figure 4 shows the probability that a job of length x hours will complete given that it is submitted during hour y of the day. Note that this is assuming that no other jobs are running at the time and should therefore be considered as an overestimate of the probability. As all computers are rebooted at 3am this leads to the diagonal cut-off within the heat map going from a 50% chance of completion to 0% in the lower right hand side of the figure. There is only one hour slot under which a

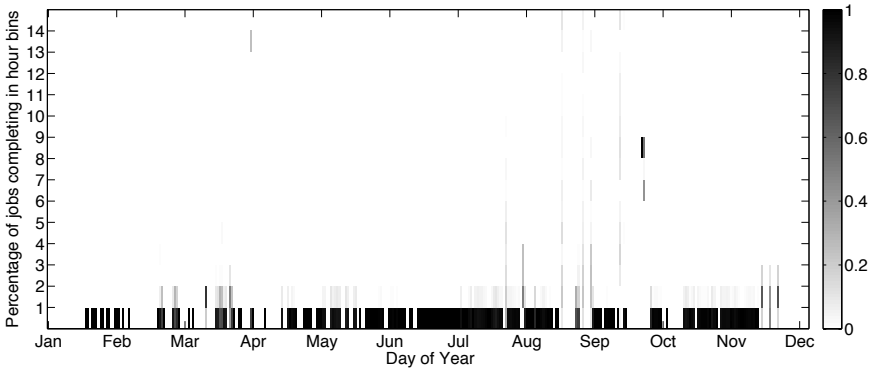

Figure 3: Breakdown of Job durations per day

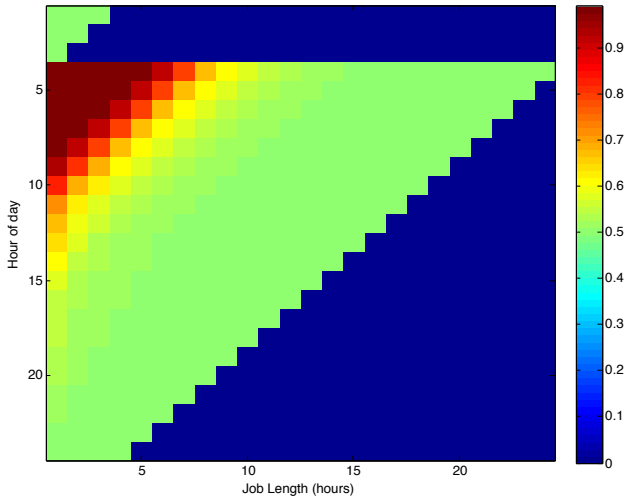

Figure 4: Probability job duration per hour of the day

24 hour job can complete – when started immediately after a computer reboot at 3am. The highest chance of short jobs completing successfully being between 3am and 8am. By using Figure 4 along with largest prior execution time for an evicted job we can determine with some degree of confidence the chances that the job will complete at the time of submission.

Although Figure 4 provides great insight into the best times to run jobs on the HTCondor system this analysis is highly specific to the system at Newcastle University and is a static snapshot of the system as it was in 2010. Ideally we would wish to produce a dynamic version of this information which could adapt to any similar volunteer computing system.

4. Reinforcement Learning Approach

Reinforcement Learning (RL) [7] is a machine learning technique in which an agent can learn without the aid of supervision a set of actions in response to a given set of states. RL has the advantage of not requiring a training data set and can adapt its behaviour to a changing environment. RL has been previously used to solve control problems such as elevator scheduling or resource allocation

within a data centre [10].

In order to use Reinforcement Learning to reduce the energy consumption for our system we use the approach of an n-armed bandit [7]. Under this assumption each action – the resource on which to run the job – is independent of all other resource selections. An extra action allows a job to remain in the queue and not receive service at this time. Each job $j \in \{1, 2, ...\}$ to be scheduled will observe the system in a given state $s \in S$. For simplicity we assume here that a job which is evicted becomes a new job within the system when it is re-scheduled. An action $a \in A$ then needs to be determined as to which resource the job should be allocated:

$$a = f(Q(s, A)), \tag{1}$$

where $Q(s, A)$ is the set of all reward values for the actions A available whilst in state s and $f()$ is the selection policy. Although we do not know $Q(s, A)$ we can estimate $Q'(s, A)$ from prior decisions and rewards and use this as an estimator:

$$Q'_j(s, A) = \{q'_j(s, a)\} \quad \forall a \in A, \tag{2}$$

and:

$$q'_j(s, a) = \overline{R_i(s, a')} \quad \forall i \leq j, a' = a, \tag{3}$$

where $R_i \in [-1, 1]$ is the reward function for job i. A value, for $R_i(s, a)$, of -1 indicating that this was the worst possible choice of action whilst $+1$ indicates the best choice of action. Before defining the reward function we need first to define the four possible outcomes when an action is applied to a job. These are:

- **Job i ran to completion on resource a:** The job i was submitted to resource a and ran to completion. In this case the action was good and we would wish to reward it well ($+1$). However, we may not wish to give this a reward of $+1$ if there exists a more energy-efficient resource on which the job could have been placed.

- **Job i was allocated to a resource and was either evicted or killed:** This is the counter-case to the first item. In this case the resource selection was bad (-1). We may however not wish to fully punish this with a value of -1 if there are other resources which have worse energy consumption which could have been chosen.

- **Job i was queued but there exists a resource a' which could have run it to completion:** In this case the job was placed into the queue but it was later determined that another resource a' which was idle at the time could have serviced the job to completion. Thus the decision to queue was a bad one (-1). In order to determine if there exists a resource a' we must track all resources which were free at the selection time and determine if any remain free until the completion time of the job. Note that we only penalise a queue action once even if there are multiple resources which could have serviced the job.

- **Job i was queued and no resource could have run it:** This is the counter-case to the one listed above. In this case no resource can be found which was free at the time of selection and remained free long enough to service the job. In this case the choice to queue was the best choice.

We can now define the reward function as follows for job i:

$$R_i(s,a) = \begin{cases} +1 - \sigma E_a & i \text{ ran to completion on resource } a \\ -1 & i \text{ was queued but there exists a resource } a' \text{ which could have run it to completion} \\ +1 & i \text{ was queued and no resource could have run it} \\ -1 + \sigma(1 - E_a) & i \text{ was allocated to a resource and was either evicted or killed,} \end{cases} \tag{4}$$

where the first term in the reward function is used to indicate that the chosen action was good or not and the second term (if present) helps to steer jobs

towards more energy efficient resources. The value $\sigma \in [0, 1]$ is the ratio of how important energy conservation is over not wasting energy through badly placed jobs and $E_a \in [-1, 1]$ is a scaled value indicating how energy efficient the selected computer is in comparison to the most and least energy efficient computers available:

$$E_a = \frac{e_a - e_b}{e_w - e_b}, \tag{5}$$

where e_a is the energy rate of the resource used, e_b the energy rate of the most power efficient computer and e_w the energy rate of the worst computer available. In all cases the active energy rate is used.

We can now define the selection policy $f()$ which is used to determine the action to perform given the prior history reward $Q'(s, A)$. We define two approaches here, those of a greedy selection and an explorative selection policy:

$$f(Q'(s, A)) = \begin{cases} max_a(Q'(s, A')) & \text{with probability 1 - } \epsilon \quad \text{(exploitative)} \\ selectRandom(A') & \text{with probability } \epsilon \quad \text{(explorative)} \end{cases} \tag{6}$$

where $A' \subset A$ is the set of all actions which are currently available. Thus A' is the set of resources which are currently free along with leaving the job in the queue. $max_a()$ selects the action a with the highest expected reward, whilst $selectRandom(A')$ will select an action uniformly at random from A'.

If we select a greedy policy then we are exploiting prior knowledge to use the action with the greatest expected reward, whilst an exploitive policy allows us to search for potentially better actions. This is particularly important due to the dynamic state of our system. Being too greedy can lead to poor energy saving as the agent will keep using sub-optimal actions, whilst being too explorative can lead to wasted energy whilst trying different actions. A careful selection of ϵ is therefore required.

4.1. Computer level approach

For the above RL approach we can define the state set as the hour during the day which the job arrives along with the maximum number of hours that the job has consumed on a previous evicted executions (if any, otherwise zero). This gives us a state space of 24^2 states as our computers reboot every day at 3am leading to a maximum execution time of 24 hours. The set of possible actions can be to allocate jobs to specific computers within the whole system along with an action to place the job into the queue. In this case there are $24^2(n+1)$ possible state-action combinations, where n is the number of computers.

4.2. Cluster level approach

The potential search space for the Computer based RL could potentially be too great and lead to drawbacks of time to compute the best action to take, memory footprint for storing the state-action combinations or the time taken for the RL algorithm to converge on a good policy. One way to alleviate these problems is to reduce the search-action space. We can exploit here the fact that sets of computers are co-located within clusters. We can keep the state space the same here and have actions which select which cluster to send a job to, or leave the job in the queue. This reduces the state-action space to $24^2(c+1)$ combinations, where c is the number of clusters. Alternatively we can increase the state space by looking at the hour within a week that the job arrives. Thus allowing the weekly patterns of the system to be taken into account. This gives a state-action space of $24 \times 168(c+1)$ combinations. Allocation of jobs to computers within a cluster is then performed at random over those computers currently available. It should be noted that a cluster which had no computers free at the time of scheduling would not have been considered.

4.3. System level approach

In this degenerative case we use RL only to select between running a job and placing it onto the queue. It can be seen as a dynamic implementation of the heat map presented in Section 3. We have 24^2 states and only two actions

(run job, queue job). If the action is to run the job then a computer is selected at random from all of those available within the system. This is presented as a quick and low-memory version of RL and to validate if computer or cluster selection RL can outperform a simple hour of day / prior job execution time policy.

4.4. Optimisation of RL approaches

Here we discuss a number of approaches taken to optimise the efficiency of the RL approaches discussed above.

4.4.1. Varying the reward history for RL

The usage patters of a volunteer computing system can show great variation and seasonal patterns – see Section 5 for a discussion of the patterns for our HTCondor system. As such taking all prior history into account when computing the expected reward can make the RL slow to respond to changes. In order to overcome this limitation we investigate the use of limiting the history that we take into account. For each state-action combination we may choose to only take the n most recent rewards into account when computing the average previous reward. This helps reduce the impact caused by events happening much earlier in the system, especially when there is significant variation within the system.

The approach of limiting the history used when computing the average reward can be extended to give higher weights to more recent rewards than rewards further into the past. Here we use a gaussian weighting of the rewards to replace equation 3:

$$q'_j(s,a) = \frac{\sum_{i=j-n}^{j-1} R_i(s,a) w_{i-j+n+1}}{\sum_{i=j-n}^{j-1} w_{i-j+n+1}}, \tag{7}$$

where w_i is the gaussian weight function:

$$w_i = e^{\frac{-8(n-i)}{n}} \tag{8}$$

and all other parameters are as defined for equation 3. The gaussian weight function is chosen to give a weight of 1 to the most recent reward and a weight

of almost zero to the nth most recent reward. All rewards prior to this are assumed to have a weight of zero and have no effect on the weighted average.

4.4.2. Vary ϵ

An alternative approach to modifying the history is to modify ϵ to make the RL approach more explorative when the rewards become less favourable. We use two complementary approaches here:

- **Initially high ϵ:** Here the initial value ϵ_1 is set high, for each state, until the first n rewards have been observed. After this it reverts back to a lower value ϵ_2. This allows the RL to be initially more explorative.

- **Vary ϵ when the new rewards diverge from the best seen so far:** Here the average of the last n rewards, for a given state, are compared with the best reward seen so far. If the ratio falls below a set threshold then the value of ϵ is increased. This allows the RL to become more explorative when the rewards move away from the best seen so far.

4.4.3. Preventing known bad actions

Under certain circumstances the set of all available actions may not be a sensible set to select from. Rather than waiting for the RL approach to learn these patterns we can instead remove these from the possible search space before selecting the action. We present here three approaches we have implemented to prevent bad actions:

- **Removing full Clusters / Computers:** At the time of action selection the current state of the HTC system is evaluated. If the RL policy is based on individual computers then the action space is reduced to only those computers which could currently accept a job and enqueueing the job. Likewise for the cluster-based RL policy only those clusters capable of running the job (and enquiring) are used. In the degenerative case – system level – if all computers are currently in use then the job will just be

enqueued. This prevents the RL from allocating jobs to resources which would not be able to handle them.

- **Favour untried actions when all tried actions are bad:** If RL is being exploitative at a given time then this can lead to bad action choices. This is especially the case at the start when only a few of the actions may have been tried and if these have all given poor rewards. In order to reduce this effect we artificially give all untried actions a reward value of zero. Thus if all previous actions from this state have lead to bad (negative) rewards then the RL approach will select an untried action as giving the best reward. As all good rewards will give a positive value this would not prevent a previously identified good action from being selected.

- **Using local system knowledge:** We can exploit here information we know about our particular HTC setup in order to remove action combinations which are known a-priori not to lead to good rewards. In the case of the HTCondor setup at Newcastle University as all the computers are rebooted at 3am there is no point in selecting an action which would mean a job is active at this time. Thus any job which has a start hour and previous (evicted) run-time which would lead to the job being active at 3am is only ever placed into the queue. This can be seen as an implementation of the harsh cut-off shown in Figure 4. Note that this does not prevent jobs from being evicted at 3am due to a reboot – when the previous run-time of the job did not indicate that it would run past 3am.

5. Experimental environment

The HTCondor installation at Newcastle University makes use of 1,359 student access computers, which were running Microsoft Windows XP in 2010. These computers were distributed around 37 *'clusters'* based in different locations around the University. Computer clusters may share the same room, with each room having its own opening hours. These hours vary between clusters

that are predominantly for teaching purposes and open during teaching hours (normally 9am till 5pm) through to 24-hour access computer clusters. The location of clusters has a significant impact on throughput of interactive users, from clusters buried deep within a particular school to those within busy thoroughfares such as the University Library.

Computers within the clusters are replaced on a five-year rolling programme with computers falling into one of three broad categories as outlined in Table 1. The University has had a policy to minimise energy consumption on all computational infrastructure for a number of years. Hence the 'Normal' computers have been chosen to be energy efficient. 'High End' computers are provisioned for courses requiring large computational and/or rendering requirements such as CAD or video editing, as such they have higher energy requirements. 'Legacy' computers pre-date the policy of purchasing energy efficient computers and are also the oldest equipment within the system. All computers within a cluster are provisioned at the same time and will contain equivalent computing resources. Thus there is a wide variance between clusters within the University but no significant variance within clusters.

Whilst we expect casual use to migrate onto user owned portable devices and virtual desktops, the demand for compute/graphic intensive workstations running high-end software is, if anything, increasing. Further, these high-end applications are unlikely to migrate to virtual desktops or user owned devices due to hardware and licensing requirements, so we expect to need to maintain

Table 1: Computer Types

Type	**Cores**	**Speed**	**Power Consumption**		
			Active	**Idle**	**Sleep**
Normal	2	~3Ghz	57W	40W	2W
High End	4	~3Ghz	114W	67W	3W
Legacy	2	~2Ghz	100-180W	50-80W	4W

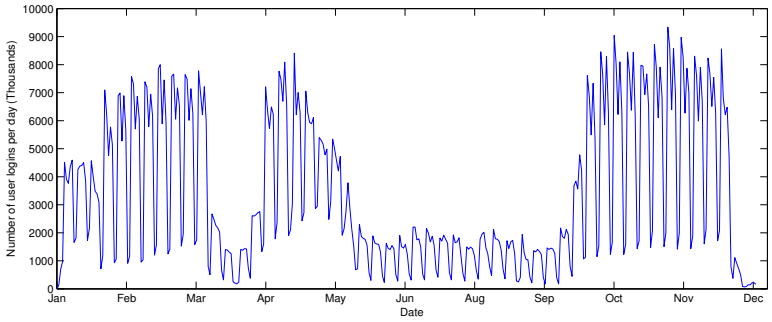

Figure 5: Interactive user logins showing seasonality

a pool of hardware that will also be useful for HTCondor for some time.

By default computers within the cluster will enter the sleep state after a given interval of inactivity. This time will depend on whether the cluster is open or not. During open hours computers will remain in the idle state for one hour before entering the sleep state whilst outside of these hours the idle interval before sleep is reduced to 15 minutes. This policy was originally trialled under Windows XP where the time for computers to resume from the shutdown state was considerable (sleep was an unreliable option for our environment). Likewise the time interval before a HTCondor job could start using a computer was set to be 15 minutes during cluster opening hours and zero minutes outside of opening hours. The latter was possible as computers would only have their states changed at these times due to HTCondor waking them up or a scheduled reboot.

We have trace logs generated from interactive user logins and HTCondor execution logs for 2010. Figure 5 illustrates the interactive logins for this period showing the high degree of seasonality within the data. It is easy to distinguish between week and weekends as well as where the three terms lie along with the vacations. This represents 1,229,820 interactive uses of the computers. It would therefore seem reasonable to expect that jobs which are run during quieter times would have a greater chance of successful completion than those run during the most busy weekdays.

Figure 6 depicts the profile for the 532,467 job submissions made to HTCondor during this period. As can be seen the job submissions follow no clearly definable pattern. Note that out of these submissions 131,909 were later killed by the original HTCondor user or the system administrator as the jobs were not completing and wasting resources. In order to handle these killed jobs the simulation assumes that these will be non-terminating jobs and will keep on submitting them to resources until the time at which the high-throughput user (or system administrator) terminates them. However, the RL approach should identify these and keep them in the queue until their termination time. It is worth noting that on Thursday 03/06/2010 there were approximately 93,000 job submissions.

Through previous work [8] we are able to wake up sleeping computers when required using *Rooster* [11]. This capability is replicated within the simulation.

5.1. Simulation Software

We have been developing a trace driven simulation model of a shared resource High Throughput Computing system, based around the HTCondor software, since 2010 [6, 8, 12]. This simulation software allows us to rapidly evaluate different policy ideas and scheduling ideas without the need to alter the live HTCondor environment or requiring lengthy deployment of a test environment. Once an idea has shown good potential within the simulation environment we

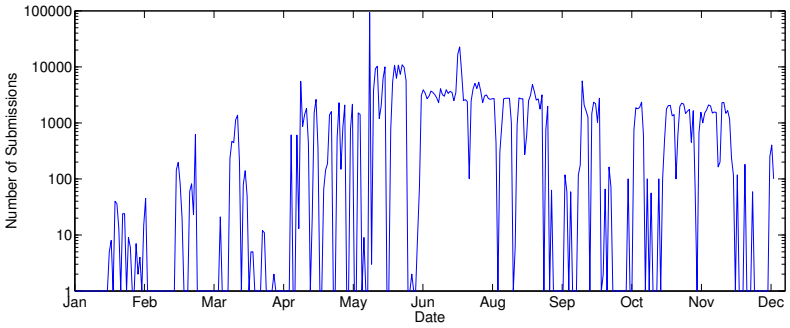

Figure 6: HTCondor job submission profile

have been able to deploy the ideas out into the real HTCondor environment [8].

The simulation software consumes three files, the first describing the policy configuration to use for the simulation, the second a trace log of user access patterns to the computers and the third file a trace log of HTCondor workload. The user trace data indicates login and logout time for the user, and the specific computer that the user occupied. In this paper we do not simulate alterations to this usage pattern. The high-throughput trace data, by contrast, contains only the time that the jobs were submitted and their duration as changes to the policy will change the computers used and the time at which the jobs complete. By interplaying these two log files under the defined policy we are able to compare different policy and scheduling decisions. Although the approach has been developed with the HTCondor system in mind it could be easily be adapted to other high-throughput clustering systems.

We have extended our cluster-based simulation for HTCondor [12] to take account of the data transfer times. The *iperf* bandwidth testing software [13] was used to compute the maximum bandwidths available between computers for different payload sizes. Although bandwidth for small (less than 1Kb) of data exceeded 100MBits/s this quickly capped out at 94.75MBits/s. It should be noted that these are maximum bandwidth potentials, real use is likely to be less. Thus these are lower estimates of transfer times.

6. Related Work

A number of scheduling approaches have adopted the use of Reinforcement Learning. Bodík *et al.* [10] proposed the use of Machine Learning, and Reinforcement Learning in particular for machine allocation within a Data Centre under defined Quality of Service requirements. Galstyan *et al.* [14] applied Reinforcement Learning in the context of resource allocation in grid environments, using Q-learning with an ε-greedy selection rule, applying the technique to a synthetic workload comprising 1000 agents and 250 resources, showing the mechanism to outperform both 'random' and 'least loaded' allocation approaches. Tesauro *et*

al. [15] proposed the Sarsa(0) approach for resource allocation in multiple server hosting environments for web applications. This was an extension to previous work [16]. Das *et al.* [17] applied Reinforcement Learning to optimise performance and energy consumption for a homogeneous group of servers, achieving 25% savings with negligible impact on SLAs. However, these approaches are focused at more short running tasks on dedicated resources without the need to deal with job eviction due to other users of higher priority.

Reinforcement Learning has also been widely used for scheduling at a lower level within task throughput systems. Bar-Hillel *et al.* [18] apply Reinforcement Learning techniques to automatically adapt the number of concurrent tasks running on a grid workstation, proposing both online and batch approaches, though the presented results were only for a small deployment. While Vengerov *et al.* [19] used RL for real-time processor core allocation. Whiteson *et al.* [20] employed Reinforcement Learning to devise a user request routing policy for multi-tier applications. Rao *et al.* [21] applied Reinforcement Learning in VCONF, an agent for dynamic reconfiguration of virtual machines. Kephart *et al.* [22] used RL to develop powercap policies for performance and power management of a single chassis of blade servers. As these approaches work at a different level to our work we see them as being complementary.

A number of Grid and Cluster level simulators exist including SimGrid [23], GridSim [24], and OptorSim [25] though these focus more at the resource selection process both within clusters and between clusters and lack the modelling of energy. More recently Cloud simulators have been proposed which are capable of modelling tradeoff between not only cost and Quality of Service, but also energy consumption. These include CloudSim [26], GreenCloud [27], and MDCSim [28]. However, these do not allow modelling of multi-use clusters with interactive user workloads.

A number of studies [29, 30] leverage resource heterogeneity and devise power management and workload distribution schemes to achieve near energy-proportional [31] cluster power profiles. Due to the long-running nature of the tasks comprising our workload, such approaches would incur significant job

overheads and reduction in overall system throughput.

Berten and Jeannot [32] performed a numerical analysis of resubmissions in a fault prone Grid environment. Their approach studies the effect of bounded and unbounded reallocation polices. However, energy consumption is not considered and tasks are assumed not to be faulty.

Checkpointing and migration [33] does not reduce task reallocation but removes the need to re-start the task after each reallocation. However, to allow checkpoint and migration the task and the environment needs to support this process, something which is currently unavailable in the Windows implementation of HTCondor which makes up the majority of the Newcastle pool. Users can 'roll' their own checkpoint and migration mechanism, however this is often a non-trivial task to perform.

Estimates of task execution times can be used as a criteria for selecting when to deploy a job to a resource. However, the use of estimates, provided by users at submission, have been widely criticised by the scheduling community for their inaccuracy [34]. With many papers reporting the majority of task taking less than 30% of their requested allocation [35, 36, 37]. This may be due to tasks misconfiguration causing immediate termination [38] but is often due to wide variation in execution times [39] – especially if the cluster is heterogeneous – or since tasks are often terminated at the end of their estimated time interval users 'pad' their estimate to increase the chance of completing. The use of estimates could be added into our system to help during the initial runs of the job.

7. Simulations and Results

Here we present the results of comparing the different Reinforcement Learning approaches we have defined in Section 4. We also evaluate the effect of the two parameters we defined within our RL algorithm – specifically ϵ and σ. Here ϵ indicates how exploitative or explorative we wish to be and σ indicates our priority towards selecting the most energy efficient computer.

We compare the different approaches and parameters through the average

overheads (measured in minutes) observed within the simulation and the energy consumed in processing the HTC jobs. We define the overhead of a job to be the difference between the true execution time of a job and the wall-clock time between job submission and job completion. For energy we will report on the total power consumed (in MWh) for high-throughput jobs in the period. We do not concern ourselves with the energy consumed via the interactive users, as this is assumed to be constant, except to say that this equates to approximately 202MWh. For the HTC energy consumed we break this down into *'good energy'* – energy expended in running jobs to completion – and *'wasted energy'* – energy wasted on jobs which are evicted or terminated by the user / administrator. It should be noted that running HTCondor jobs on different hardware may lead to variations in execution times. It would however be difficult to determine how this would be affected without knowing whether the particular job was memory- CPU- or IO-dominant. As such the simulations ignore this effect and assume the job will require the same time to execute.

As a point of reference we present here the overheads and energy consumed by our simulation acting under the policy which is currently in place at Newcastle University – computers power managing themselves as described in Section 5 and using the default HTCondor resource selection policy (effectively a random resource selection policy). The average overhead within the system was 13.5 minutes and a total energy consumption of 121MWh. The total energy consumption comprised of 37.4MWh of good energy and 83.6MWh of wasted energy.

7.1. Effect of ϵ and σ

We present here only results for the RL cluster approach as these are indicative for all other cases. Figure 7 shows the effect of ϵ and σ on the average overheads observed. There is no apparent impact on overheads due to σ, this is consistent with the RL and our assumption that the execution time for a job remains constant irrespective of the computer used.

There is a significant impact on overheads for small values of ϵ (<0.1). This

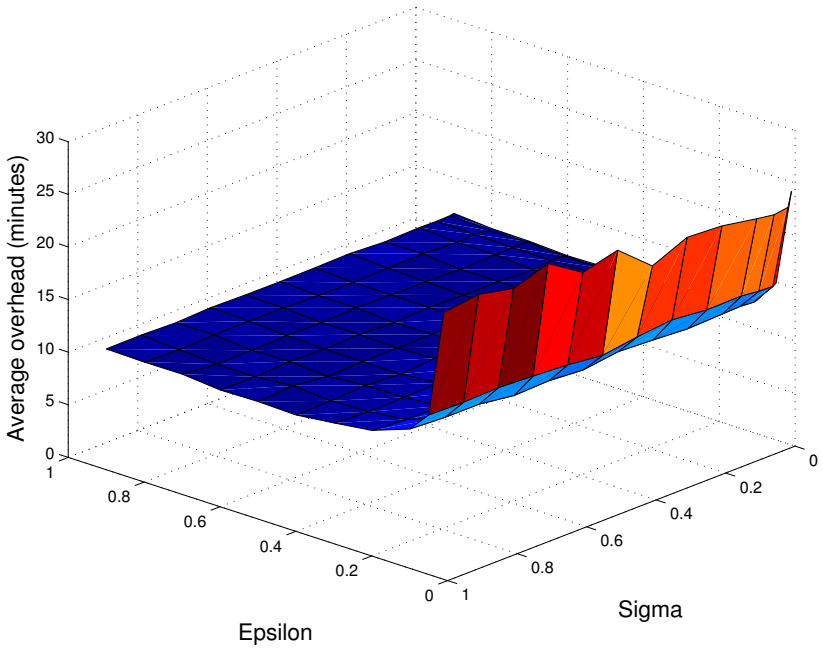

Figure 7: Average overhead on RL jobs

is a consequence of the RL approach being too exploitative and not identifying the best resources to use. This is bourn out within the simulation by resources being selected which fail to run the job to completion requiring resubmissions. However, once ϵ is greater than 0.1 there is little if any effect on the overheads seen within the system.

Looking at the total energy consumed for the HTCondor workload (Figure 8) reducing the value of ϵ reduces the energy consumed. However, unlike the overheads the effect is more graduated over the range of ϵ starting from $\epsilon = 5$. With energy consumptions ranging from ˜114MWh down to ˜57MWh. This equates to an overall energy saving between 6 and 53% in comparison to the current non-RL approach taken. Thus choosing a low value of ϵ would seem to give the most energy efficient solution. However, given that values of ϵ less than 0.1 lead to an increase in overheads if the desire is to maintain the same overheads then ϵ should be set to 0.1. This will give a total energy consumption

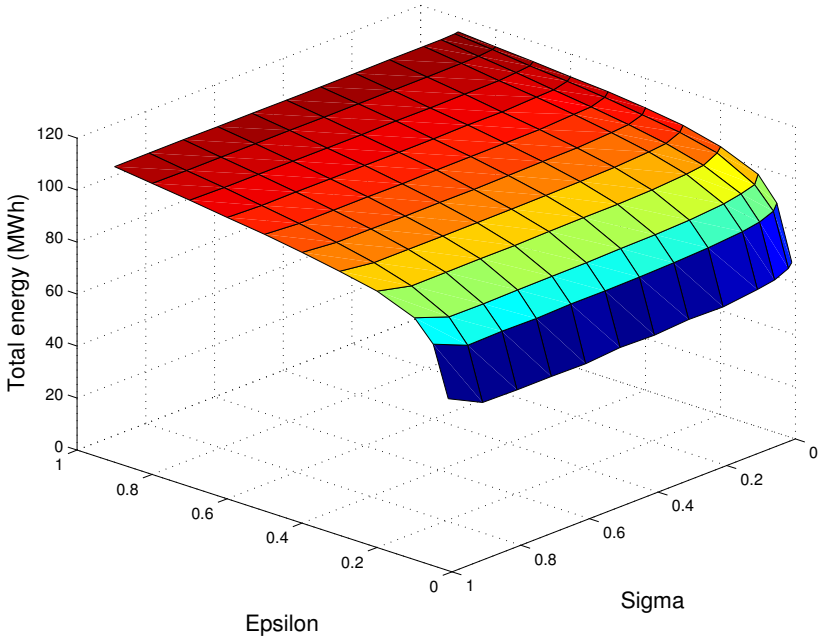

Figure 8: Total Energy consumed for RL jobs

of ˜87MWh, which is a saving of 28%.

Although not clearly visible within Figure 8 σ has an effect on the energy consumed by the system. This effect is a consequence of the savings which can be made on the good energy used within the system and can be seen a little easier in Figure 9. The effect is most pronounced for small ϵ where the difference is ˜7MWh (6%) whilst at large values of ϵ this falls to just ˜1MWh (0.8%). At an ϵ value of 0.1 the difference is still ˜7MWh. Therefore selecting a large value of σ, with the minimum value being seen with $\sigma = 0.8$, would give the largest energy saving.

The effect on useful energy by ϵ ranges between ˜4MWh and ˜10MWh (3-8%) with an ϵ value of 0.1 only increasing the energy consumed by ˜1MWh.

By contrast the wasted energy (Figure 10) shows no impact from varying σ. Although σ does play a role in equation 4 it is apparent that the wasted energy equation is dominated by the two other cases which do not have a σ component.

The wasted energy (Figure 10) is significantly impacted by the value of ϵ

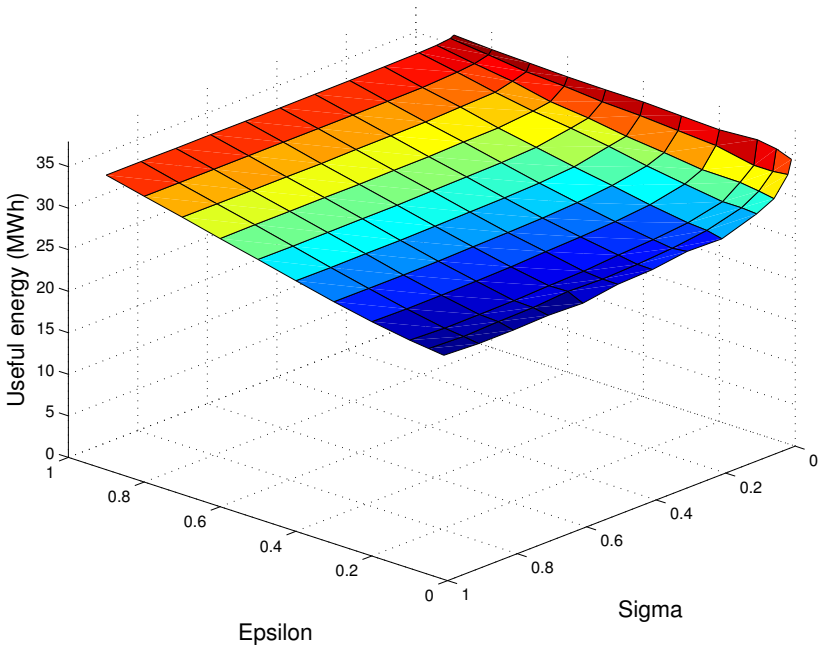

Figure 9: Useful Energy consumed for RL jobs

with low values of ϵ exhibiting the lowest wasted energy values. This provides a potential energy saving of ˜47MWh (39%).

Thus if the overall concern is to minimise overheads choosing $\epsilon = 0.1$ would give the best value. Whist if the overall concern is to minimise energy consumption then $\epsilon = 0.01$ and $\sigma = 0.8$ would provide a reduction of ˜57MWh (53%). Given that we wish to maintain the overhead at a reasonable level choosing $\epsilon = 0.1$ and $\sigma = 0.8$ would reduce energy consumption by ˜34MWh (28%).

Figure 11 illustrates the relationship between overhead and total energy consumed when we vary ϵ and σ. The different colours within the graph each represent a different value of ϵ whilst the spread of points represents the impact of σ. The major impact here is from ϵ whilst the the effect of σ is much less – though increasing as ϵ decreases. Choosing lower values of ϵ will minimise energy consumption, though at the expense of increasing overheads. Whilst increasing ϵ will increase the energy consumed and decrease the overheads. However,

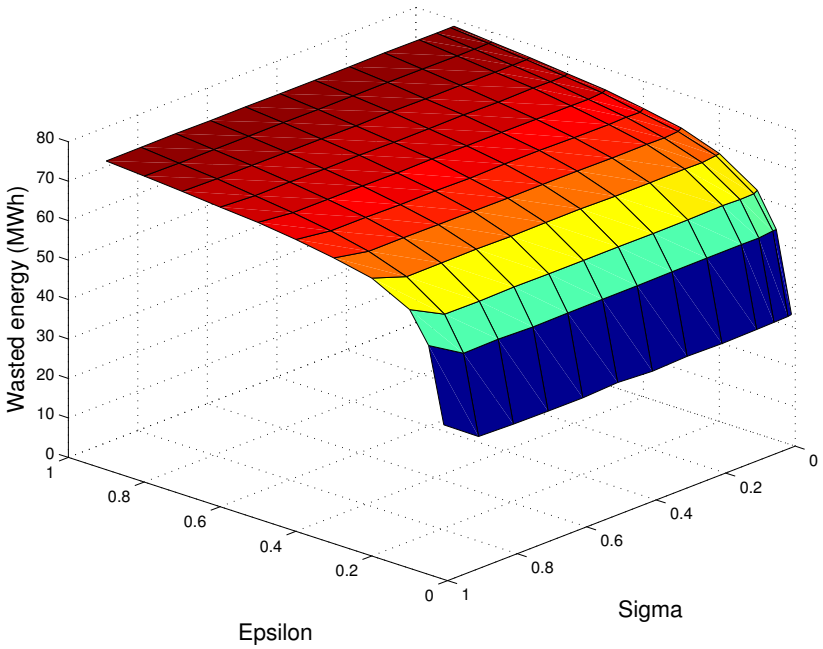

Figure 10: Wasted Energy consumed for RL jobs

increasing ϵ beyond 0.6 will increase the overheads whilst still increasing the energy consumption providing no benefit for overheads nor energy. Thus the maximum value of ϵ which should be selected is 0.6.

7.2. Comparison of different RL approaches

Here we compare the four identified RL strategies: Computer, Cluster, Cluster Week and System. These are discussed in Sections 4.1, 4.2, 4.2 and 4.3 respectively. For each approach we select three combinations of ϵ and σ which give the lowest overhead, lowest energy and the closest overhead in comparison to our non-RL approach.

Figure 12 shows the overheads observed for each of these approaches. The minimum overheads are observed for the Computer RL approach (9.9 minutes) though these are closely followed by cluster week and cluster, only being 0.4 and 0.7 minutes longer on average respectively. The system level approach is unable to compete being some six minutes longer than the best RL approach and

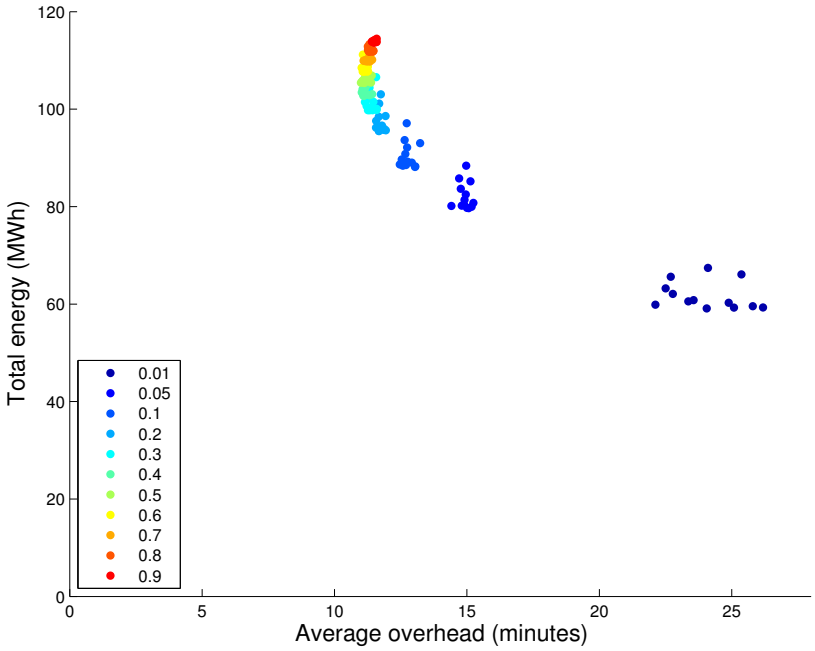

Figure 11: Overhead and Energy compared

some 2.3 minutes slower than the current non-RL approach. However, none of these approaches show good energy reductions (Figure 13, where LO is Lowest Overheads, LE is Lowest Energy and RC is matching the overhead from our non-RL selection policy). The largest reduction is for the cluster week approach which achieves a ˜21MWh (17%) reduction. This is achieved through the lowest values of good energy and wasted energy.

For the case where we wish to minimise energy consumption the cluster approach is the best to select, this reduces the consumed energy by ˜64MWh (53%). However, this does lead to the highest overheads (30.7 minutes). By contrast the cluster week approach has the lowest overheads for any of the lowest energy choices (13 minutes). However this is at the expense of raising energy consumption by ˜24MWh (20%).

All approaches, apart from the system level approach, are capable of matching the non-RL policy. The cluster level approach manages the best energy

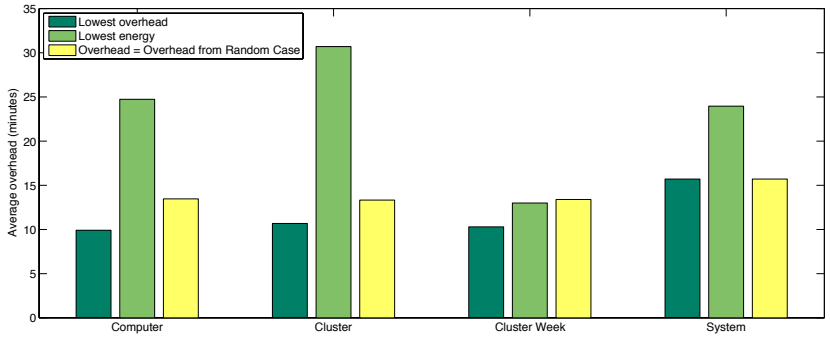

Figure 12: Comparison of the overheads for the different RL approaches

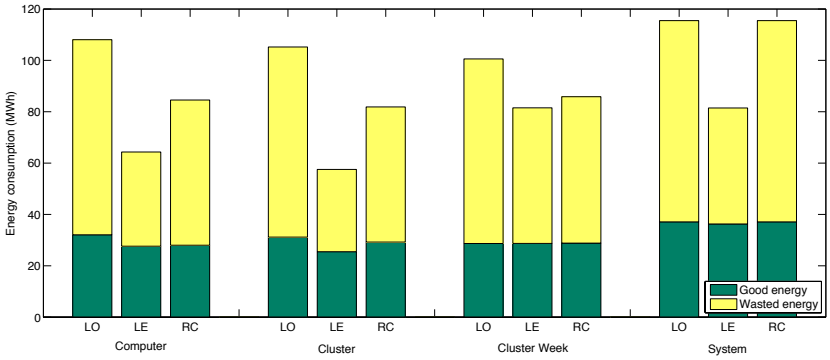

Figure 13: Comparison of the energy consumed for the different RL approaches

consumption here reducing it by ˜36MWh (30%).

In all cases the dominant energy usage is on wasted work. However, this is still significantly reduced in comparison to the non-RL approach, ranging from ˜5MWh (4%) for the system level approach to ˜51MWh (42%) for the best energy cluster approach. By contrast the good energy reduction varies between ˜12MWh (10%) for the cluster approach and ˜0.3MWh (0.2%) for the system level approach.

Thus if our primary concern is saving energy we should adopt a cluster approach. Whilst for minimising overheads we should choose the computer level approach. If we wish to maintain the overheads seen in our current system we should use the cluster level approach. In general the system level approach

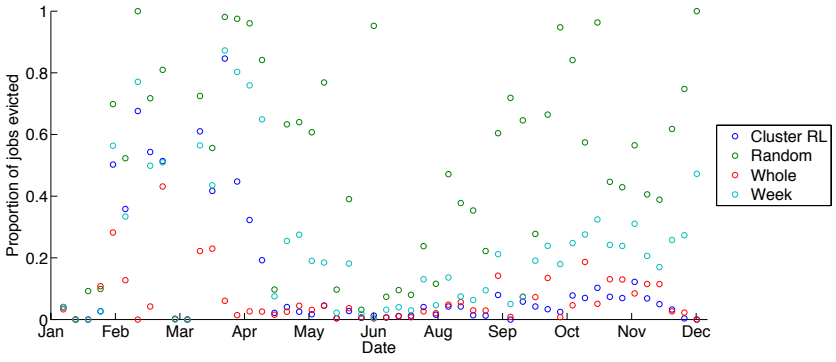

Figure 14: Proportion of jobs which are evicted each week

seems to have little benefit. This is likely due to the fact that it lacks the fidelity required for the RL approach to learn the underlying patterns. The cluster level approach appears to be in general the best approach as it comes out on top for two of the three scenarios. For the other scenario (minimising overheads) it is only 0.7 minutes slower (with better energy consumption). Given that the state-action storage for this approach is also significantly smaller, leading to quicker searching, it would appear to be the best approach.

7.3. RL performance

Figure 14 shows the proportion of jobs launched in a given week which end up being evicted. As expected the number of evictions for the non-RL case (random) fluctuates widely during the year and still reaches high values at the end of the year. Whilst all RL approaches all start high early in the year and then become smaller as the year progresses. Both whole system and cluster-based RL manage the lowest number of evictions in the latter part of the year. Whilst the cluster-week RL approach fails to remain as low after September. This is likely to be a consequence of the fact that the larger state space receives fewer rewards per action thus taking longer to adapt to the changes in the underlying system.

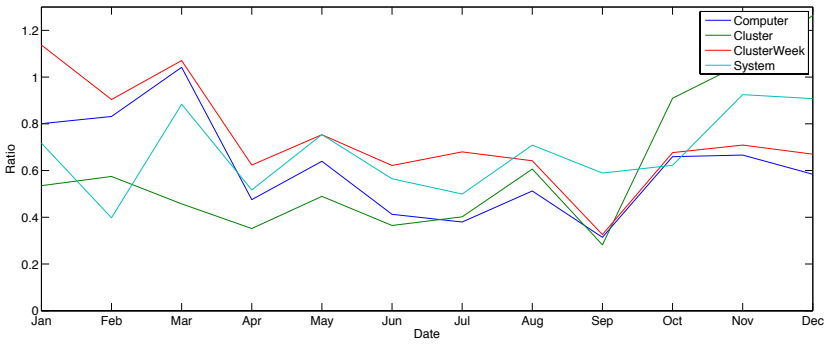

Figure 15: Ratio of energy used over the year when optimising for Energy

7.4. Comparison of energy savings

Here we investigate how the different approaches compare over the course of the year. We define here the ratio between energy consumed per month for the non-RL policy in comparison with the energy used by each of the RL approaches.

Figure 15 shows how the four different RL approaches compare over the entire of 2010 when comparing the most energy efficient combination. The cluster approach has the best energy saving, as is bourn out in the previous results, however it looses out from September onwards. This is due to the low value of ϵ (0.01) meaning that the system is slow to react to changes. The computer approach is much better at reacting to change despite having the same value of ϵ. This is most likely due to the fact that as the reward value is computed as the average of all previous reward values (R) then as the number of reward values increases then the effect of each individual R value on the mean becomes less. As the computer approach has a larger state-action space than the cluster approach the number of values for any given state-action combination is likely to be less.

A comparison of the different approaches in the case of keeping the overheads the same as the non-RL approach is presented in Figure 16. Here the cluster level approach works best, having $\epsilon = 0.05$ is enough to make it reactive to changes in the system but still maintain a good exploitation of the already

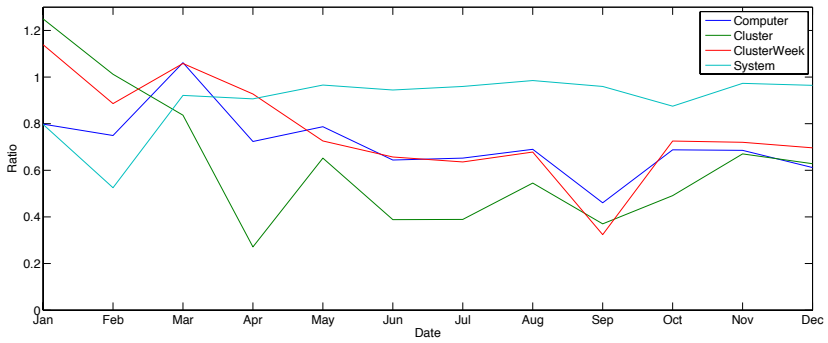

Figure 16: Ratio of energy used over the year when keeping overheads the same

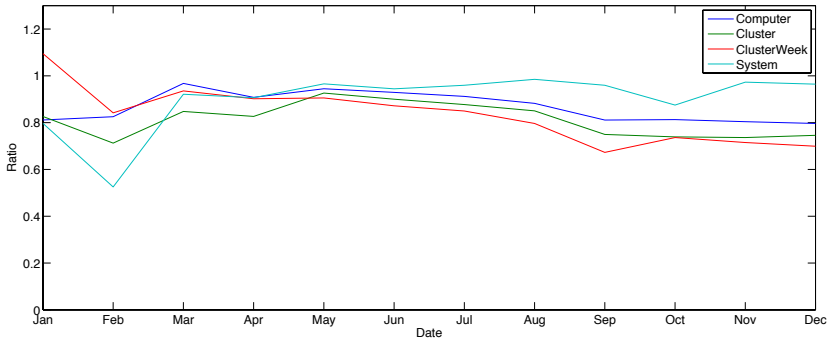

Figure 17: Ratio of energy used over the year when optimising for overheads

determined actions. The computer and cluster week approaches are close to the cluster approach. The computer approach also having $\epsilon = 0.05$ whilst the cluster week approach having $\epsilon = 0.01$. The two approaches are close though when the underlying interactive logins increase in September-October the cluster week approach is affected more. The system level approach is not optimal across the year, again suggesting that this lacks the fidelity required for the RL approach.

The case where we optimise for overheads is shown in Figure 17. As we are not taking the energy consumption of the system into account here the energy saving is small (only around 20%). Thus showing that when optimising for overheads we are sacrificing most if not all of the potential energy savings.

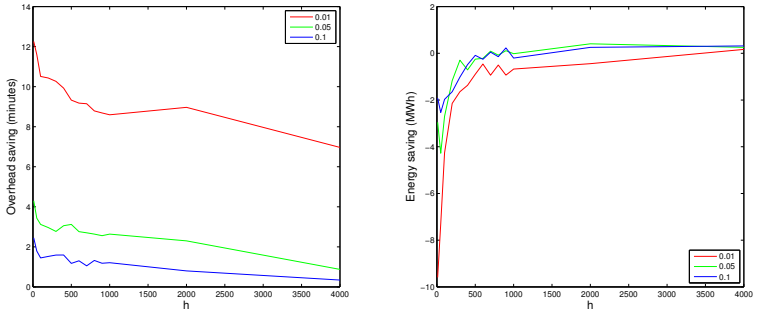

Figure 18: Effect of limiting History of Rewards

7.5. Optimisation of RL

7.5.1. Varying the reward history for RL

Figures 18 and 19 show the impact of limiting the history used when determining the expected reward for a given action. In each case we vary the size of the history h and observe its impact on the overheads observed and the energy consumed. In all cases the overhead (energy) saved is in comparison to the case when the entire history is taken into account and all lines will tend towards one as h tends to infinity – the case of taking all history into account.

For both the un-weighted average (Figure 18) and the gaussian weighted average (Figure 19) using this approach will save overhead whilst increasing energy consumption. Apart from very small values of h the un-weighted average increases the energy consumption less than the gaussian weighted average and in general seems to offer a greater saving of overhead. Thus the un-weighted average would appear to be the most sensible option.

7.5.2. Initially high ϵ

Having a higher value of ϵ at the start of the simulation has shown no statistical difference in the energy consumed or the overheads observed by jobs. This is consistent for n rewards between 50 and 1,000. We expect this to be a consequence of the fact that jobs tend to arrive in bursts which quickly exceed n for certain states. As these then become the dominant states in the system

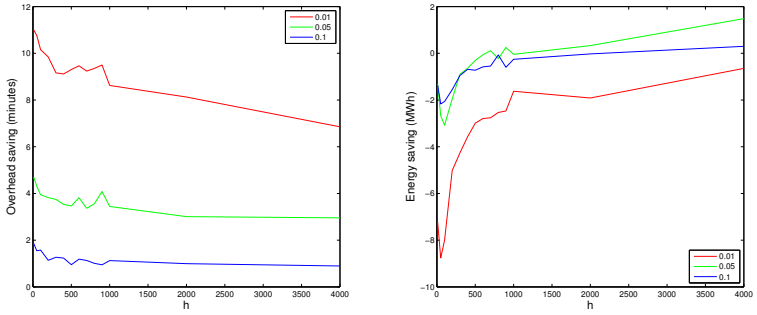

Figure 19: Effect of limiting History of Rewards using a gaussian function

the added benefit of a more explorative approach for other states is lost.

7.5.3. Vary ϵ when the new rewards diverge from the best seen so far

Here we have investigated increasing the value of ϵ by 0.1 when the ratio of rewards falls below a threshold. However, this has not shown any significant statistical change to either the energy consumed by the system or the overheads observed by the user. This is likely to be a consequence of the low utilisation of our HTC system allowing jobs to receive service quite quickly.

7.5.4. Using local system knowledge

The effect of using local knowledge on the system – preventing jobs from starting which would run over the reboot time – shows no statistical advantage. Jobs which would have been run before the reboot time would still have received service once the reboot had taken place and thus will in general receive the same overhead time. Whilst for the energy saved through not running the jobs before the reboot will be small as the system seems to learn that this is a bad option quickly.

8. Conclusions

In this paper we have shown that for a multi-use volunteer computing system the time of the day at which a job is submitted directly influences the chances

of the job completing without being evicted due to an interactive user with higher priority or a computer reboot. We argue that a static analysis of such a volunteer computing system will not be able to adapt to the changing nature of the interactive users.

We therefore develop four Reinforcement Learning approaches, based on a computer-by-computer action approach, an approach where actions are based on a collection of computers co-located within a cluster, an adapted cluster approach which takes the day of the week into account and a course grained Reinforcement Learning approach which only selects between allocating work to a computer and queueing the work up for future deployment.

Through simulation results we demonstrate that the cluster based approach, with $\epsilon = 0.1$ and $\sigma = 0.8$ gives the best results. We go further to show that such a Reinforcement Learning approach could save between 30% and 53% of the energy used by the volunteer computing system depending on whether we wish to maintain the overheads on work execution times currently observed.

The choice of an averaging of reward values within the Reinforcement Learning approach seems to lead to a reduction in sensitivity to change as the total number of pieces of work increases. This would suggest that discarding old reward values would help when running this approach for a long time.

The approach of Reinforcement Learning is a powerful mechanism for identifying complex patterns within a system and allowing decisions to be made over these patterns. We anticipate that this approach could be effective within other parts of our system and are currently investigating the use of Reinforcement Learning in the selection of times to perform checkpointing of jobs.

References

[1] M. Litzkow, M. Livney, M. W. Mutka, Condor-a hunter of idle workstations, in: 8th International Conference on Distributed Computing Systems, 1998, pp. 104–111.

[2] D. P. Anderson, Public Computing: Reconnecting People to Science, Pre-

sented at the Conference on Shared Knowledge and the Web, Residencia de Estudiantes, Madrid, Spain.

[3] M. Litzkow, T. Tannenbaum, J. Basney, M. Livny, Checkpoint and migration of UNIX processes in the Condor distributed processing system, Computer Sciences Department, University of Wisconsin, 1997.

[4] M. Forshaw, A. S. McGough, Energy-efficient checkpointing in high-throughput cycle-stealing distributed systems, Electronic Notes in Theoretical Computer Science accepted for.

[5] M. Forshaw, A. S. McGough, N. Thomas, On energy-efficient checkpointing in high-throughput cycle-stealing distributed systems, in: 3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS) 2014, 2014.

[6] A. McGough, M. Forshaw, C. Gerrard, S. Wheater, Reducing the number of miscreant tasks executions in a multi-use cluster, in: Cloud and Green Computing (CGC), 2012 Second International Conference on, 2012, pp. 296–303. doi:10.1109/CGC.2012.111.

[7] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, A Bradford book, Bradford Book, 1998.

[8] A. S. McGough, P. Robinson, C. Gerrard, P. Haldane, S. Hamlander, D. Sharples, D. Swan, S. Wheater, Intelligent power management over large clusters, in: International Conference on Green Computing and Communications (GreenCom2010), 2010.

[9] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd and Toshiba Corporation, ACPI Specification, http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf.

[10] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, D. Patterson, Statistical machine learning makes automatic control practical for internet datacenters, in: Proceedings of the 2009 Conference on Hot Topics in Cloud

Computing, HotCloud'09, USENIX Association, Berkeley, CA, USA, 2009. URL http://dl.acm.org/citation.cfm?id=1855533.1855545

[11] The HTCondor Team, HTCondor manual, http://research.cs.wisc.edu/htcondor/manual/, May 2013, University of Wisconsin.

[12] A. S. McGough, C. Gerrard, J. Noble, P. Robinson, S. Wheater, Analysis of power-saving techniques over a large multi-use cluster, in: International Conference on Cloud and Green Computing (CGC2011), 2011.

[13] Sourceforge project, The iperf project, http://iperf.sourceforge.net/.

[14] A. Galstyan, K. Czajkowski, K. Lerman, Resource Allocation in the Grid Using Reinforcement Learning, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3, IEEE Computer Society, 2004, pp. 1314–1315.

[15] G. Tesauro, N. K. Jong, R. Das, M. N. Bennani, A hybrid reinforcement learning approach to autonomic resource allocation, in: Autonomic Computing, 2006. ICAC'06. IEEE International Conference on, IEEE, 2006, pp. 65–73.

[16] G. Tesauro, et al., Online resource allocation using decompositional reinforcement learning, in: AAAI, Vol. 5, 2005, pp. 886–891.

[17] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, H. Chan, Autonomic multi-agent management of power and performance in data centers, in: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 107–114.

[18] A. Bar-Hillel, A. Di-Nur, L. Ein-Dor, R. Gilad-Bachrach, Y. Ittach, Workstation capacity tuning using reinforcement learning, in: Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on, IEEE, 2007, pp. 1–11.

[19] D. Vengerov, N. Iakovlev, A reinforcement learning framework for dynamic resource allocation: First results., in: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on, IEEE, 2005, pp. 339–340.

[20] S. Whiteson, P. Stone, Adaptive job routing and scheduling, Engineering Applications of Artificial Intelligence 17 (7) (2004) 855–869.

[21] J. Rao, X. Bu, C.-Z. Xu, L. Wang, G. Yin, Vconf: a reinforcement learning approach to virtual machines auto-configuration, in: Proceedings of the 6th international conference on Autonomic computing, ACM, 2009, pp. 137–146.

[22] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. L. Rawson III, C. Lefurgy, Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs., in: ICAC, Vol. 7, 2007, p. 24.

[23] A. Legrand, L. Marchal, Scheduling distributed applications: The simgrid simulation framework, in: In Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid, 2003, pp. 138–145.

[24] R. Buyya, M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience 14 (13) (2002) 1175–1220.

[25] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, F. Zini, Optorsim - a grid simulator for studying dynamic data replication strategies, International Journal of High Performance Computing Applications.

[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50. doi:10.1002/spe.995. URL http://dx.doi.org/10.1002/spe.995

[27] D. Kliazovich, P. Bouvry, Y. Audzevich, S. U. Khan, Greencloud: A packet-level simulator of energy-aware cloud computing data centers, in: GLOBECOM, 2010, pp. 1–5.

[28] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, C. Das, Mdcsim: A multi-tier data center simulation, platform, in: Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009, pp. 1–9. doi:10.1109/CLUSTR.2009.5289159.

[29] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, R. Katz, Napsac: Design and implementation of a power-proportional web cluster, ACM SIGCOMM computer communication review 41 (1) (2011) 102–108.

[30] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, X. Zhu, Delivering energy proportionality with non energy-proportional systems–optimizing the ensemble (2008).

[31] L. A. Barroso, U. Holzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33–37.

[32] V. Berten, E. Jeannot, Modeling Resubmission in Unreliable Grids: the Bottom-Up Approach, in: Seventh International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks - heteroPar'09, Delft, Netherlands, 2009.

[33] M. Litzkow, T. Tannenbaum, J. Basney, M. Livny, Checkpoint and migration of UNIX processes in the Condor distributed processing system, Computer Sciences Department, University of Wisconsin, 1997.

[34] C. Bailey Lee, Y. Schwartzman, J. Hardy, A. Snavely, Are user runtime estimates inherently inaccurate?, in: Job Scheduling Strategies for Parallel Processing, Springer, 2005, pp. 253–263.

[35] W. Cirne, F. Berman, A comprehensive model of the supercomputer workload, in: Proceedings of the Workload Characterization, 2001. WWC-4.

2001 IEEE International Workshop, WWC '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 140–148.

[36] W. A. Ward, Jr., C. L. Mahood, J. E. West, Scheduling jobs on parallel systems using a relaxed backfill strategy, in: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP '02, Springer-Verlag, London, UK, UK, 2002, pp. 88–102.

[37] S.-H. Chiang, A. C. Arpaci-Dusseau, M. K. Vernon, The impact of more accurate requested runtimes on production job scheduling performance, in: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP '02, Springer-Verlag, London, UK, UK, 2002, pp. 103–127.

[38] A. Mu'alem, D. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling, Parallel and Distributed Systems, IEEE Transactions on 12 (6) (2001) 529 –543.

[39] J. P. Jones, B. Nitzberg, Scheduling for parallel supercomputing: A historical perspective of achievable utilization, in: Proceedings of the Job Scheduling Strategies for Parallel Processing, IPPS/SPDP '99/JSSPP '99, Springer-Verlag, London, UK, UK, 1999, pp. 1–16.